B.2 pytest

pytest is a Python library used to run tests for your code. In this section, we'll describe how to write tests that are automatically discovered and run by pytest, how to actually run pytest in your code, and some tips and tricks for making the most of pytest.

How do you write a pytest test?

A **test** in pytest is a Python function whose name starts with test_. Inside a test function, we use assert statements to verify expected values or behaviours of a function.

For example:

```
# This is the function to test
def has_more_trues(booleans: list) -> bool:
    """Return whether booleans contains more True values than False values.

    >>> has_more_trues([True, False, True])
    True
    >>> has_more_trues([True, False, False])
    False
    """
    # Function body omitted


# This the test
def test_mixture_one_more_true() -> None:
    """Test has_more_trues on a list with a mixture of True and False,
    with one more True than False.
    """
    assert has_more_trues([True, False, True])
```

A single test can have multiple assert statements, although it is generally recommended to separate each assert statement into a separate test. A single Python file can have multiple tests; when pytest is run on a file, it (by default) runs all the tests in that file.

Running pytest

The simplest way of running pytest is to add the following `if __name__ == '__main__'` block to the bottom of a test file:

```
if __name__ == '__main__':
    import pytest
    pytest.main()
```

When you run this file, `pytest.main` will run all test functions in the file. *Note*: by default, `pytest.main` actually searches through *all* Python files in the current directory whose name starts with `test_` or ends with `_test`, which can be a bit surprising. So our practice will be to explicitly pass in the name of the current test file to `pytest.main`, wrapped in a list:

```
# If we're in a file test_my_file.py

if __name__ == '__main__':
    import pytest
    pytest.main(['test_my_file.py'])
```

Testing for an exceptions

It is possible to write a `pytest` test that checks whether a function raises a specific error. To do so, use `pytest.raises`, which takes an error type as an argument, inside a `with` statement. Here is an example

```
import pytest


def add_one(n):
    return n + 1


def test_add_one_type_error():
    """Test add_one when given a non-numeric argument."""
    with pytest.raises(TypeError):
        add_one('hello')
```

Options for `pytest.main`

`pytest.main` takes a list of strings as an argument because users can add options (as strings) to modify `pytest`'s default behaviour when running tests. The format for this is `pytest.main([<option1>, <option2>, ...])`.

Here are some useful options:

- `'<filename>'`: as we saw above, adding a filename restricts `pytest` to only running the tests in that Python file.

- `'<filename>::<test_name>'`: restrict `pytest` to run a specific test in the given file (e.g., `'test_my_file.py::test_1'`)
- `'-x'`: stop running tests after the first failure (by default, `pytest` runs all tests, regardless of the number of failures)
- `'--pdb'`: start the Python debugger when a test fails

References

For the full documentation for the `pytest` library, check out https://docs.pytest.org/en/latest/.